

An Efficient Quasi-physical Algorithm for Packing Equal Circles in a Circular Container

Kun He, Hui Ye¹, Zhengli Wang², Tian Xie

School of Computer Science, Huazhong University of Science and Technology, Wuhan 430074, China

Abstract

This paper addresses the equal circle packing problem, and proposes an efficient quasi-physical algorithm (EQPA). EQPA is based on an improved BFGS algorithm and a new basin hopping strategy. Starting from a random initial pattern, we use the modified BFGS algorithm to reach a local minimum pattern. The modified BFGS algorithm fully utilizes the neighborhood information and considerably speeds up the running time of the gradient descent process, and the efficiency is more apparent for larger scale instances. The new basin-hopping strategy is to shrink the container size when yielding a local minimum pattern. Experimental results indicate that the new basin-hopping strategy is very efficient, especially for a type of pattern with comparatively dense packing in the center and sparse packing around the bounding. We test EQPA on the instances of $n = 1, 2, \dots, 320$, and obtain 66 new patterns which have smaller container sizes than the current best-known results reported in literature.

Keywords: Circle packing, global optimization, BFGS, basin hopping

1. Introduction

1.1. General introduction for CPP

The circle packing problem (CPP) is concerned with arranging n circles in a container without overlap, which is of great interest to industry and

¹Corresponding author. Email: yehui20080414@gmail.com

²Corresponding author. Email: 498195342@qq.com

academia. CPP is encountered in a variety of real world applications, including apparel, naval, automobile, aerospace and facility layout (Castillo et al. (2008)). CPP has been proven to be NP-hard (Demaine et al. (2010)), and it is hard to get an exact solution in polynomial time even for some specific instances. Researchers resort to heuristic methods in order to solve the problem efficiently, which fall into two categories: construction methods and optimization methods.

The construction methods can be described as packing circles one by one using some rules. The first type of rules fixes the container radius and only concerns where to feasibly arrange the circles in the container. This type of algorithms include the Max Hole Degree (MHD) (Huang et al. (2003)) based algorithms, the self look-ahead search strategy (Huang et al. (2005), Huang et al. (2006)), the Pruned-Enriched-Rosenbluth Method (PERM) (Lü and Huang (2008)), and the beam search algorithm (Akeb et al. (2009)). The other type of rules adjusts the container radius during the construction procedures, including the series of Best Local Position (BLP) (Hifi and M'Hallah (2004), Hifi and MHallah (2006), Hifi and MHallah (2007), Akeb and Hifi (2010)) and the hybrid beam search looking-ahead algorithm (Akeb and Hifi (2010)).

The optimization methods don't obtain a good solution directly, but improve the solution based on an ordinary initial solution. The great variety of optimization methods can be further classified into the quasi-physical, quasi-human algorithm (Wang et al. (2002)), the Tabu search and simulated annealing hybrid approach (Zhang and Deng (2005)), the population basin hopping algorithm (Addis et al. (2008b)), simulated annealing based algo-

rithms (Müller et al. (2009)), formulation search space algorithm (Lopez and Beasley (2013)), iterated local search based algorithms (Fu et al. (2013), Ye et al. (2013)), etc. There are two new algorithms published in 2015 that yield excellent results in 2015: the iterated Tabu search and variable neighborhood descent algorithm (Zeng et al. (2015)) and an evolutionary computation-based solution to the circle packing problem (Flores et al. (2015)).

1.2. Mathematics methods for ECPP

As a classical type of CPP, packing equal circles in a circular container, denoted as Equal circles packing problem (ECPP), is still difficult in the field of mathematics. In the early stage of the study, the value of n is relatively small, and researchers used the method of mathematical analysis, not only to find the optimal layout, but to provide proofs on the optimality. Kravitz (Kravitz (1967)), the first scholar working on ECPP, gave the layout for $n = 2, 3, \dots, 19$ with the container radius, but provided no proof of optimality. Graham (Graham (1968)) proved the optimality for $n = 2, 3, 4, 5, 6, 7$. Pirl (Pirl (1969)) proved the optimality for $n = 2, 3, 4, 5, 6, 7, 8, 9, 10$, and provided the layout of $n = 11, 12, \dots, 19$ at the same time. Goldberg (Goldberg (1971)) improved the layout for $n = 14, 16, 17$ that Pirl provided, and Goldberg also provided the layout for $n = 20$ for the first time. Ries (Reis (1975)) improved the layout for $n = 17$ based on Pirl's research, and first gave the layout for $n = 21, 22, 23, 24, 25$. Melissen (Melissen (1994)) proved the layout configuration optimality for $n = 11$, and Fodor (Fodor (1999), Fodor (2000), Fodor (2003)) proved the optimality for $n = 12, 13, 19$. Therefore, only the optimality for $n = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 19$ has been proved so far.

1.3. Heuristics for ECPP

1.3.1. Landmark for ECPP Heuristics

When n is relatively large, it is very difficult to find the optimal layout and prove the optimality. Heuristic algorithms for CPP can be very efficient to find optimal or suboptimal layouts. Although the heuristics may not guarantee the theoretical optimality, they can find some layout where the container radius is very close to the theoretical minimum.

Graham (Graham et al. (1998)) did some early work and proposed two heuristic methods. The first method is the repulsion forces. It transforms ECPP to a problem of finding the minimum on $\sum_{1 \leq i < j \leq n} (\frac{\lambda}{\|s_i - s_j\|})^m$. $S = s_1, s_2, \dots, s_n$ is the set of points in the container, λ is the zoom factor, and m is a large positive integer. After defining the objective function, we can use some existing methods (such as gradient descent method) to find the layout with the local minimum value. The other method is the billiards simulation. This is a quasi-physical method by regarding the circle items are as billiards. This algorithm starts with a smaller radius and gives the initial movement direction for each billiard. Then a series of collision motion occurs in the circular container. During the process of movement, the authors slowly increase the size for all billiards. By repeatedly running the algorithm, it is possible to find the global optimal solution. By the comprehensive use of repulsion forces and billiards simulation, Graham found the near-optimal layout for $n = 25, 26, \dots, 65$.

There are many followup work based on heuristics. Here highlight several landmark works. Akiyama (Akiyama et al. (2003)) used a greedy method to find a local optimal solution to ECPP. The algorithm continuously im-

proves the current layout by randomly moving one circle until the moving reaches an iteration limit (e.g. 300000). By repeatedly running the greedy method, Akiyama found more dense layouts for $n = 70, 73, 75, 77, 78, 79, 80$. Grosso (Grosso et al. (2010)) assumed that ECPP has the characteristic of “funneling landscape”, and used a monotone hopping strategy to look for “funnel bottom”. In order to solve the funnel problem, they used the population hopping strategy to enhance the diversity of the layout. They found a number of better layout schemes for $66 \leq n \leq 100$. Wenqi Huang and Tao Ye (Huang and Ye (2011)) proposed a global optimization algorithm using a quasi physical model. They proposed two new quasi physical strategies and found 63 better layouts among the 200 instances of $n = 1, 2, \dots, 199, 200$.

1.3.2. Main methods for ECPP

There are two key issues to solve the ECPP. First, how can we optimize an random or given layout so that it is more likely to reach the local optimum layout? Second, when we reach a local optimal layout which is not feasible, that is, there exists overlapping among some circles, we need to use some strategy to jump out of the local minimum layout and reach a new layout that inherits the advantages of the previous local optimal. Then we could continue to use the local optimization method to reach another local minimum, and we wish to eventually obtain an optimal or near-optimal layout.

For the local optimization methods, the repulsion forces and billiards simulation of Graham (Graham et al. (1998)), the monotone hopping strategy of Grosso (Grosso et al. (2010)), and the elastic force movement of Wenqi Huang and Tao Ye (Huang and Ye (2011)) in the above description all fall into this category. There are also other good methods, like TAMSASS-PECS

method (Szabó et al. (2005)), nonlinear optimization method (Birgin et al. (2005)) and reformulation descent algorithm (Mladenović et al. (2005)). Each of these algorithms has its own advantages, due to the different number of circles and the different container shapes (square, circle, rectangle, or polygon).

There are diverse methods for the Basin-hopping strategy. For example, the small random perturbation method (Addis et al. (2008a)) formed a new layout by moving several circles in the local optimal layout to some random places. However, due to the purely randomness, this method may destroy the holistic heredity. Wenqi Huang and Tao Ye (Huang and Ye (2011)) considered three kinds of forces, elastic force, attractive force and repulsive force, to promote the entire layout to a new form. They used three parameters c_1 , c_2 and *steps* to control the strength of the attractive force, the strength of the repulsive force and the duration time of the abrupt movement. Zeng et al proposed another strategy for moving random circles to vacant places in the container (Zeng et al. (2015)). By dividing the entire container into square grids, they regarded the vacant point with large vacant degree as the insertion point, which can improve the current layout to a certain extent.

1.4. *Our work*

We propose an Efficient Quasi-physical Algorithm (EQPA) for solving ECPP. Through the establishment of a physical model, we look for the minimum value of the objective function using the classical quasi Newton method, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm, but we improve the efficiency by fully utilizing the neighborhood structure of the circles. And we propose a new Basin-hopping strategy by shrinking the radius of the con-

tainer. We iteratively apply the modified sBFGS algorithm to achieve a new layout after a certain number of continuous optimization iterations, and apply the Basin-hopping strategy to jump out of the local minimum. Experiments on 320($n = 1, 2, \dots, 320$) ECPP instances demonstrate the effectiveness of the proposed method.

2. Problem formulation

The problem of packing equal circles in a circle can be described as packing n unit size circle items into a circular container. We need to ensure no overlaps between any two circles and the radius of the container R is minimized. We set a Cartesian coordinate system with its origin $(0,0)$ located at the container center, as shown in Figure 1, and locate the center of circle i ($i = 1, 2, \dots, n$) at coordinate (x_i, y_i) . We want to find a layout configuration $X = (x_1, y_1, \dots, x_n, y_n)$ such that:

$$\begin{aligned} & \text{minimize} && R \\ & \text{s.t.} && (1) \quad \sqrt{x_i^2 + y_i^2} \leq R - 1 \\ & && (2) \quad \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq 2 \end{aligned}$$

where $i, j = 1, 2, \dots, n$ and $i \neq j$.

3. Algorithm description

3.1. Physical model

Similar to previous quasi-physical methods (Ye et al. (2013), Huang and Zhan (1982), He et al. (2013), He et al. (2015)), we regard the circle items

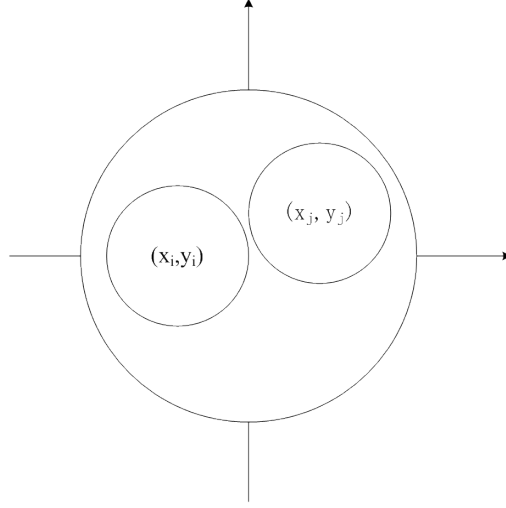


Figure 1: The coordinate system.

as elastic disks and imagine the container as a rigid circular container. If an elastic object embeds itself in another object, it will be deformed and the deformation will cause some elastic potential energy. Based on the Cartesian coordinate system, we have the following definitions.

Definition 1(*Overlapping Depth*). There are two kinds of overlaps, disk-disk overlap and disk-container overlap. The *overlapping depth* between disk i and the container is d_{oi} :

$$d_{oi} = \begin{cases} \sqrt{x_i^2 + y_i^2} + 1 - R, & \text{if } \sqrt{x_i^2 + y_i^2} + 1 - R > 0 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

and the *overlapping depth* between disk i and disk j ($i \neq j$) is d_{ij} :

$$d_{ij} = \begin{cases} 2 - \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, & \text{if } \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} < 2 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Definition 2(*Elastic Energy*). According to the elastic mechanics, the elastic

potential energy between two smoothy elastic objects is proportional to the square of the embedded depth. The elastic energy between disk i and j ($j \neq i$) is defined as:

$$U_{ij} = d_{ij}^2 \quad (3)$$

And the elastic energy U_i of disk i is defined as:

$$U_i = \sum_{j=0, j \neq i}^n U_{ij} \quad (4)$$

The total elastic energy $U(X)$ for the whole system $X = \{x_1, y_1, \dots, x_n, y_n\}$ is defined as:

$$U = \sum_{i=0}^n U_i \quad (5)$$

3.2. The BFGS continuous optimization algorithm

With the physical model established, the potential energy function $U(X)$ is defined and has the following properties:

- (1) For all $X \in (-\infty, +\infty)^{2N}$, $U(X) \geq 0$;
- (2) X is a feasible packing pattern if and only if $U(X) = 0$.

Thus, the packing problem is transformed into the optimization problem of finding the minimum value on function $U(X)$. We use the classical quasi Newton method, the BFGS algorithm, to do the continuous optimization to find local minimums.

The basic idea of the BFGS algorithm is as follows. From a current layout configuration X_k , we construct the approximate Hessian matrix H_k , using the information of objective function U and gradient g_k . Thus we can get a search direction d_k and search length λ_k , and generate a new point

X_{k+1} , we continue the process iteratively until reaching a minimum layout configuration.

The calculation of the gradient g_k is defined by Eq. (6), and the search step length λ_k is defined by Eq. (7). λ_k is a real number to take the minimum value of the elastic energy U at step k . X_k corresponds to a $2n$ – coordinate vector, and d_k is the search direction in step k .

$$g_k = \left(\frac{dU_k}{dx_1}, \frac{dU_k}{dy_1}, \dots, \frac{dU_k}{dx_n}, \frac{dU_k}{dy_n} \right) \quad (6)$$

$$\lambda_k = \arg \min_{\lambda \in R} U(x_k + \lambda d_k) \quad (7)$$

Hessian matrix is a symmetric matrix composed of the second derivatives of $U(X_k)$, and H is an approximate Hessian matrix of size $2n$ by $2n$. It is updated as Eq. (8). I is the unit matrix, s_k is the change quantity of X_k , and y_k is the change quantity of g_k . The pseudo code of the BFGS algorithm is as described in Algorithm 1.

$$H_{k+1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k} \right) H_k \left(I - \frac{y_k s_k^T}{y_k^T s_k} \right) + \frac{s_k s_k^T}{y_k^T s_k} \quad (8)$$

3.3. The Modified BFGS Algorithm

In the BFGS algorithm, when we use Eq. (5) and Eq. (6) to calculate U and g_k , for each circle i , we need to calculate its distance to each of the other $n - 1$ circles in order to judge whether the current circle has some overlaps. In the actual procedure, however, a circle usually intersects with approximately six other circles when there are a little overlaps. Ideally, if 3 equal circles are tangent to each other, the 3 lines connecting the pairwise circle centers construct an equilateral triangle. The included angle between

Algorithm 1 The BFGS algorithm

INPUT:

An initial packing layout configuration X

OUTPUT:

A locally optimal packing layout X^*

- 1: $H_0 \leftarrow I$; // H_0 is the initial approximate Hessian matrix, and I is the unit matrix.
 - 2: $k \leftarrow 0$; // k is the iteration step.
 - 3: **while** $k \leq \text{MaxIterations}$ **do**
 - 4: calculate the gradient g_k by Eq. (6)
 - 5: calculate the search direction $d_k \leftarrow -H_k * g_k$
 - 6: calculate the search step length λ_k by Eq. (7)
 - 7: $s_k \leftarrow \lambda_k * d_k$, $X_{k+1} \leftarrow X_k + s_k$
 - 8: **if** $U < 10^{-20}$ or $\|g_k\| < 10^{-10}$ **then**
 - 9: **return** the current layout X^* ;
 - 10: **end if**
 - 11: $y_k \leftarrow g_{k+1} - g_k$;
 - 12: $H_{k+1} \leftarrow (I - \frac{s_k y_k^T}{y_k^T s_k}) H_k (I - \frac{y_k s_k^T}{y_k^T s_k}) + \frac{s_k s_k^T}{y_k^T s_k}$;
 - 13: $k \leftarrow k + 1$;
 - 14: **end while**
-

two lines is 60. As the maximum degree of the central angle in a circle is 360, that is 6 times as much as 60. So there are at most six other circles tangent to a circle if there exists no overlap.

As illustrated in Figure 2, there will be little changes for the adjacency relationship of the circles after certain iterations of the BFGS Algorithm. So during the calculation procedure, for each circle in the system, we record

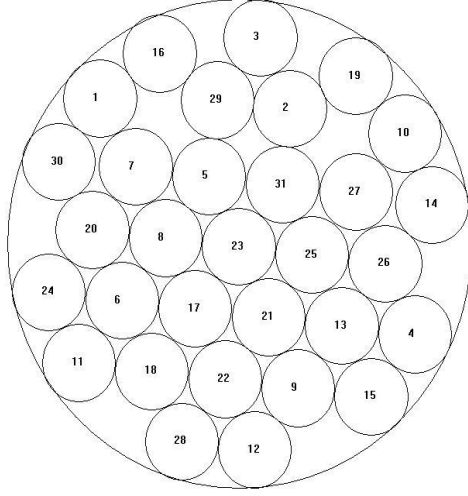


Figure 2: The local minimum layout when $n = 31$.

its overlapping circles at present and possible overlapping circles in the near future, and form an adjacency list. Specifically, for a circle i located at (x_i, y_i) , the container is regarded as an adjacency object if Eq. (9) holds. And a circle j located at (x_j, y_j) is regarded as an adjacency object if Eq. (10) holds. We set $d_1 = 1, d_2 = 1$, which means an object (circles or container) is considered as adjacent if its distance to the current circle is no greater than 1.

$$\sqrt{x_i^2 + y_i^2} + 1 - R \leq d_1 \quad (9)$$

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - 2 \leq d_2 \quad (10)$$

We only check the overlapping status for circles in the adjacency list in the modified BFGS algorithm. And we will update the adjacency list for each circle after running the modified BFGS for l iterations (we set $l = 10$ in the experiments).

We experimentally test the effectiveness of the modified BFGS algorithm on 20 instances, $n = 10, 20, \dots, 200$. In the test, we set the radius of the con-

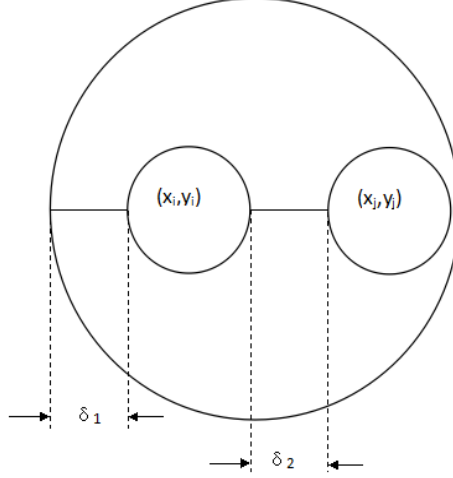


Figure 3: Distance d_1 between circle i and the container, and distance d_2 between circles i and j .

tainer as the currently known minimum radius, and record the time starting from a random initial layout to a local minimum layout. We calculate each instance for 1000 times, record the average running time T to reach the local minimum, and compare with the original BFGS algorithm. The comparison result is shown in Figure 4. With the increasement on scale n , the average time T of the two methods both increases. But the time of the BFGS algorithm increases much faster. In the case of $n = 200$, the time of the BFGS algorithm is about 4 times as much as the modified BFGS algorithm.

When using Eq. (5) to compute U and Eq. (6) to compute g_k , we only consider the adjacency circles, the time complexity reduces from $O(N^2)$ to $O(N)$. But the time complexity is still $O(N^2)$, as we need to update the approximate Hessian matrix which is n by n scale. Although the total time complexity is still $O(N^2)$, the computational efficiency shows great improvement for the local optimization, and the improvement is very obvious for

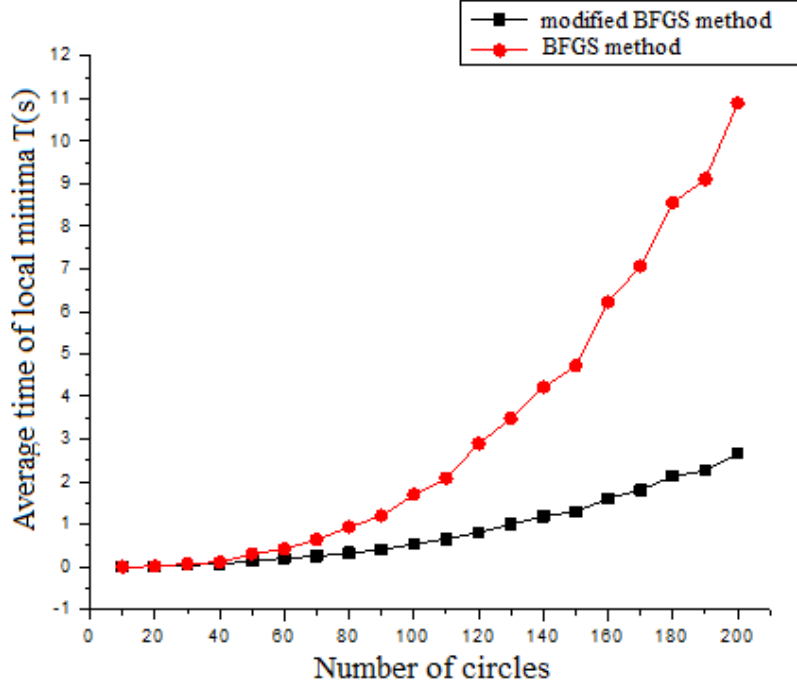


Figure 4: Improvement on the average minimum time of the modified BFGS method.

instances whose n is relatively large (such as $n \geq 200$), as shown in Figure 4.

3.4. Basin-hopping procedure

When we reach a local minimum by the modified BFGS, we shrink the container radius as the Basin-hopping strategy. We fix the original coordinate of each circle and reduce the container radius by a factor of β ($0 < \beta < 1$): $R_0 = \beta R_0$. β is defined as $\beta = 0.3 + 0.035m$ and m varies from 0, 1, 2, to 19 such that β varies from 0.30 to 0.965 to generate diverse layouts.

We then run the modified BFGS algorithm for h iterations (h is a random number ranging from 50 to 100) to reach a new layout. Figure 5 represents the layout before and after the Basin-hopping procedure. The pseudo-code

of the Basin-hopping procedure is as Algorithm 2.

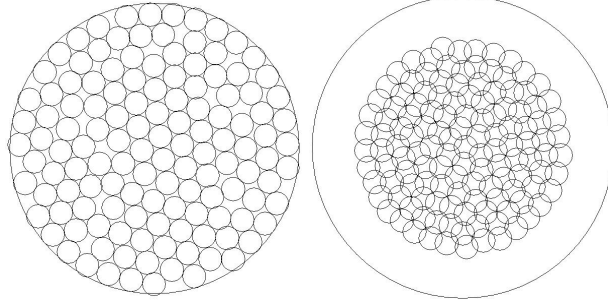


Figure 5: The layout before and after the Basin-hopping procedure when $n = 130$.

Algorithm 2 Basin-hopping procedure

INPUT:

A local minimum layout (X, R_0) .

OUTPUT:

20 new patterns.

- 1: $R \leftarrow R_0$;
 - 2: **for** $m \leftarrow 0$ to 19 **do**
 - 3: $\beta \leftarrow 0.3 + 0.035m$;
 - 4: $R \leftarrow \beta R_0$;
 - 5: $h \leftarrow$ a random integer in $[50, 100]$;
 - 6: run the modified $BFGS(R, X)$ for h iterations to get a new layout X_m
 - 7: **end for**
 - 8: $R_0 \leftarrow R$;
 - 9: **return** the 20 new patterns.
-

We experimentally test the effectiveness of the Basin-hopping procedure. Compared the Basin-hopping procedure with random initialize method, we test the instances for $n = 50, 51, \dots, 70$. In the experiments, the radius of

the container is set as the current known minimum radius. We compared the number of iterative steps which are needed to find the feasible layout using the modified BFGS algorithm.

The calculation results are shown in Figure 6. It shows that, in the cases of $n = 53, 54, 55, 66, 67, 68, 69$, the Basin-hopping procedure is apparently better. In the case of $n = 70$, the random initial method is better. In other cases, the effectiveness of two methods are almost the same. On the average, the shrinking strategy exhibits better performance.

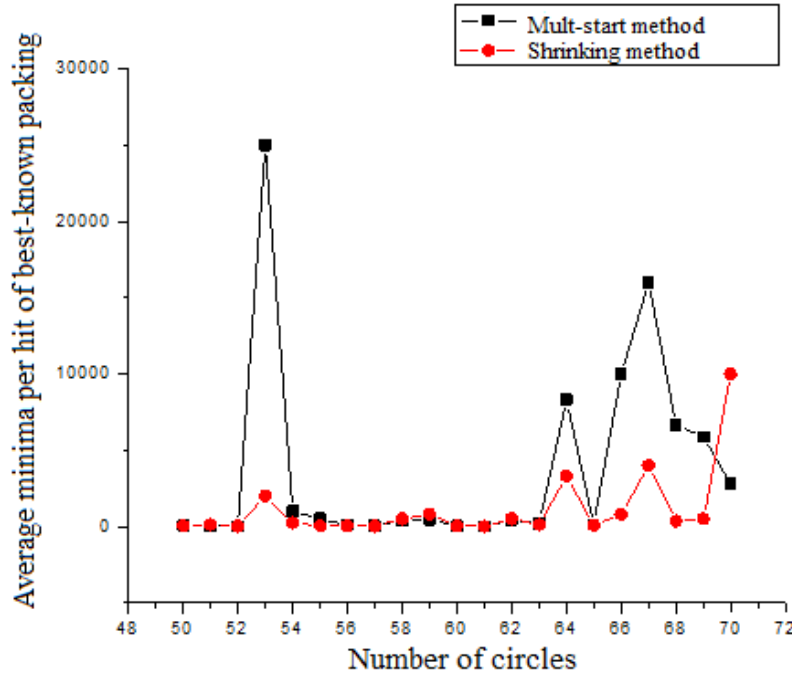


Figure 6: Comparison of the shrinking Basin-hopping procedure with random initialize method.

In the shrinking Basin-hopping procedure, the container radius decreases dramatically. After several times of iteration using the modified BFGS al-

gorithm, all circles contract to the container center. Then we recover the container radius and there is a large vacant space in the outer layer of the container. In the iterative process of the modified BFGS, all circles move outwards as a whole. While the outer space is large and the inner space is small, the result of the final movement is that there are closely inner circles and sparse outer circles. Therefore, the Basin-hopping procedure has better influence for the instances with dense inner packing and sparse outer packing, which are the most cases for the near-optimal or dense packing. We show two typical examples with the above characteristics in Figure 7.

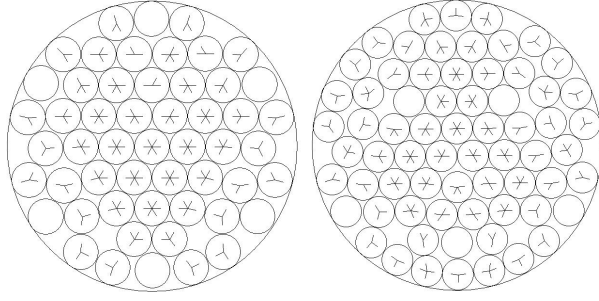


Figure 7: The global minimum layouts of $n = 53$ and $n = 68$.

Combining the modified BFGS algorithm with the Basin-hopping procedure, we present the global search algorithm in Algorithm 3.

3.5. The container adjustment procedure

For the new layout after the Basin-hopping, the radius and location of the container are constant, and circles are centralized around the center of the container. We then continue the optimization with the modified BFGS algorithm and hopefully to reach a better layout. During this procedure, viewing from the layout picture, most circles are moving from the container's

Algorithm 3 The Global Search Algorithm

INPUT:

R_0 .

OUTPUT:

A feasible or stuck layout (X, R_0) .

```
1: randomly generate an initial layout  $X$ ;  
2: while  $U > 10^{-20}$  and the timeout limit is not reached do  
3:   run modified BFGS( $X, R_0$ );  
4:   if  $U < 10^{-20}$  then  
5:     record the current layout;  
6:     EXIT;  
7:   end if  
8:   run Basin-hopping  $(X, R_0)$  generate 20 new layouts and store them in the  
   array  $C$ ;  
9:   for  $i = 1$  to 20 do  
10:     $C[i] \leftarrow \text{modifiedBFGS}(C[i])$ ;  
11:    if  $U < 10^{-20}$  then  
12:      goto step 5;  
13:    end if  
14:  end for  
15:  select the best configuration from  $C$  as  $X$ ;  
16:  goto step 8;  
17: end while
```

center to the border. If the terminate condition $U < 10^{-20}$ is satisfied while the circles haven't been tangent to the border of the container, it is obviously that the radius of the container can still be decreased until the circles is

Algorithm 4 Container Adjustment Algorithm

INPUT:

Current best minimum layout $(X, R_{bestknown})$.

OUTPUT:

New feasible layout (X, R_{min}) .

```
1:  $R \leftarrow R_{bestknown}$ ;
2:  $i \leftarrow -1$ ;
3: repeat
4:    $R_{upperbound} \leftarrow R$ ;
5:    $i \leftarrow i + 1$ ;
6:    $R \leftarrow R_{upperbound} - 10^{-10} \times 10^i$ ;
7:   run the modified BFGS algorithm( $X, R$ );
8: until  $(R, X)$  is not feasible;
9:  $R_{lowerbound} \leftarrow R$ ;
10: repeat
11:    $R_{min} \leftarrow (R_{upperbound} + R_{lowerbound})/2$ ;
12:   run the modified BFGS algorithm( $X, R_{min}$ );
13:   if  $(R_{min}, X)$  is feasible then
14:      $R_{upperbound} \leftarrow R_{min}$ ;
15:   else
16:      $R_{lowerbound} \leftarrow R_{min}$ ;
17:   end if
18: until  $|R_{upperbound} - R_{lowerbound}| \leq 10^{-10}$ .
```

tangent to the border of the container. So we use the container adjustment strategy to decrease the radius of the container.

We start with the current known records $R_{bestknown}$ as the container's initial radius. If the global search algorithm returns an overall minimum layout, we hope to maintain the relative locations of the circles and further decrease the radius of the container to get a more compact layout. Here we use the dichotomy method to find the minimal radius R_{min} of the container from the current layout. We update $R_{bestknown}$ to R_{min} , and use the global search algorithm for a new round of search.

For the instance of $n = 130$, $R_{bestknown} = 12.6023189367$. The global search algorithm finds the global minimum layout as shown in the left portion of Figure 8. After the container adjustment procedure, the container radius changes to $R_{min} = 12.6017746136$, which is reduced by about 6×10^{-4} , as shown in the right portion of Figure 8.

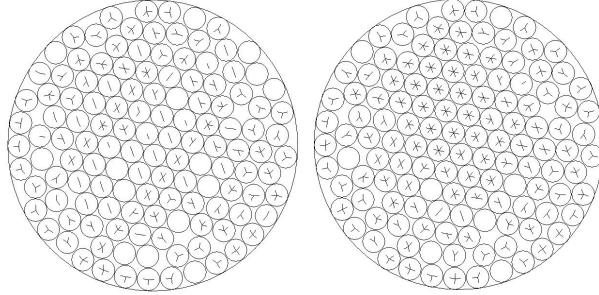


Figure 8: The layout before and after the container adjustment procedure for $n = 130$.

4. Experimental results

4.1. Experimental environment

We implement EQPA with C++ programming language. The IDE is Visual C++ 6.0. We run EQPA on the Ali cloud platform (<http://www.aliyun.com>). The platform has an 8 core processor and an 8GB memory. Its OS is Windows Server 2012, and the experiment is completed without any parallelism technique.

4.2. Computational efficiency and robustness

The first set of experiments is to test whether the algorithm can robustly find the current best-known layout. For the first 100 instances ($n = 1, 2, \dots, 100$), the radius of the container is set to the current known minimum (downloaded from the official website <http://www.packomania.com>, 20 June 2016). We randomly locate n circles in the container and then call the global search algorithm. Then we get the 20 new layouts for the next iterative. Once the program find a feasible layout or the running time is more than 4 hours, the calculation will terminate. In order to verify the robustness of the algorithm, each instance is calculated 10 times.

The results show that for the first 100 instances ($n = 1, 2, \dots, 100$), except $n = 82, 100$, EQPA found 98 currently best-known layouts. For $n \leq 49$, the algorithm can always stably find the current best-known layout. In these 49 instances, the average computation time is at most 720s.

The corresponding experimental statistics for $n = 50, 51, \dots, 100$ are as shown in Table 1. The first column is the number of circles. The second column corresponds to the currently known minimum radius of the container.

The third column is the hit count among the 10 times calculation, i.e., the number of feasible layout found by EQPA within 4 hours. And the forth column is the average computational time in seconds.

n	$R_{bestknown}$	Hit Count	Time(seconds)	n	$R_{bestknown}$	Hit Count	Time(seconds)
50	7.9475152747	10/10	34	76	9.7295968021	10/10	464
51	8.0275069524	10/10	21	77	9.7989119245	10/10	147
52	8.0847171906	10/10	6	78	9.8577098998	3/10	4276
53	8.1795828268	10/10	1043	79	9.9050634676	10/10	5464
54	8.2039823834	10/10	33	80	9.9681518131	5/10	5145
55	8.2111025509	10/10	27	81	10.0108642412	7/10	5446
56	8.3835299225	10/10	21	82	10.0508242234	0/10	-
57	8.4471846534	10/10	7	83	10.1168578751	7/10	4821
58	8.5245537701	10/10	71	84	10.1495308672	8/10	5364
59	8.5924999593	10/10	67	85	10.1631114658	10/10	377
60	8.6462198454	10/10	13	86	10.2987010531	2/10	5867
61	8.6612975755	10/10	4	87	10.3632085050	10/10	1148
62	8.8297654089	10/10	861	88	10.4323376927	10/10	4866
63	8.8923515375	10/10	92	89	10.5004918145	5/10	7751
64	8.9619711084	10/10	2872	90	10.5460691779	10/10	588
65	9.0173973232	10/10	34	91	10.5667722335	10/10	40
66	9.0962794269	10/10	2404	92	10.6846458479	4/10	4992
67	9.1689718817	10/10	7492	93	10.7333526002	3/10	6092
68	9.2297737467	10/10	3252	94	10.7780321602	8/10	5552
69	9.2697612666	10/10	713	95	10.8402050215	5/10	6056
70	9.3456531940	10/10	1752	96	10.8832027597	4/10	5824
71	9.4157968968	10/10	916	97	10.9385901100	1/10	11540
72	9.4738908567	10/10	113	98	10.9793831282	9/10	2593
73	9.5403461521	2/10	5571	99	11.0331411514	9/10	3445
74	9.5892327643	6/10	5343	100	11.0821497243	0/10	-
75	9.6720296319	8/10	6671				

Table 1: The bestknown container radius, hit count and computation time for 10 rounds on $n = 50, 51, \dots, 100$.

The second set of experiments is to test whether EQPA can find a better layout than the current best layout. For the first 320 instances ($n = 12, \dots, 320$), the radius of the container is set to the currently known minimum (downloaded from official website <http://www.packomania.com>, 20 June 2016). We randomly locate n circles in the container and then run the global optimization algorithm. Once we find the minimum layout, we

apply the container adjustment procedure to decrease the radius of the container, and run the algorithm again, until we can not reduce the container size with the time limit.

The results show that for 66 of the 320 instances, EQPA found better layouts than the current best instances. Table 2 shows the specific improvements. In Table 2 and 3, R_0 is current known minimum container radius, R^* is smaller container radius obtained by EQPA, $R_0 - R^*$ is the improvement degree. The improvements are in the range of 10^{-7} to 10^{-2} , and most improvements are in 10^{-3} or 10^{-4} .

Figure 9 further illustrates some new layouts obtained by EQPA.

n	R_0	R^*	$R_0 - R^*$	n	R_0	R^*	$R_0 - R^*$
126	12.417463955	12.417144417	10^{-4}	194	15.249753585	15.241428667	10^{-2}
128	12.50231007	12.502222199	10^{-4}	197	15.368975294	15.36851843	10^{-4}
130	12.602318936	12.601774612	10^{-3}	198	15.391207855	15.390410683	10^{-3}
137	12.914725416	12.914711247	10^{-5}	204	15.652649998	15.644738394	10^{-2}
138	12.962702608	12.961304417	10^{-3}	205	15.709043665	15.695538521	10^{-2}
140	13.061097215	13.060696617	10^{-4}	207	15.770271663	15.770190573	10^{-4}
156	13.719600039	13.718404613	10^{-3}	213	15.970256294	15.969855988	10^{-4}
160	13.920538614	13.920451761	10^{-4}	214	16.01876322	16.018386421	10^{-4}
163	14.069620216	14.067635971	10^{-3}	216	16.087407652	16.087017095	10^{-4}
177	14.617194154	14.614803186	10^{-3}	217	16.119370348	16.118237367	10^{-3}
178	14.658814223	14.655927628	10^{-3}	221	16.261873984	16.261873763	10^{-7}
179	14.702293364	14.699982808	10^{-3}	222	16.299696226	16.299162417	10^{-3}
180	14.742878035	14.739035484	10^{-3}	224	16.37189924	16.369591221	10^{-3}
183	14.869399059	14.865788757	10^{-3}	225	16.409054279	16.408410528	10^{-3}
188	15.028782734	15.028750763	10^{-5}	226	16.450019501	16.449824611	10^{-4}

Table 2: Improvements in the case of $n = 126, 127, \dots, 226$.

n	R_0	R^*	$R_0 - R^*$	n	R_0	R^*	$R_0 - R^*$
227	16.494400009	16.489753739	10^{-2}	261	17.634862232	17.634766878	10^{-4}
228	16.52734087	16.527071189	10^{-4}	277	18.142486517	18.139629995	10^{-3}
229	16.566377981	16.564350195	10^{-3}	278	18.189804499	18.187675017	10^{-3}
230	16.596430072	16.596246697	10^{-4}	279	18.224068629	18.221572693	10^{-3}
231	16.640078326	16.631031907	10^{-2}	281	18.284504843	18.281539686	10^{-3}
242	16.962132986	16.961287246	10^{-3}	282	18.315754345	18.309334921	10^{-2}
243	17.004065249	17.001947599	10^{-3}	286	18.434315259	18.434157792	10^{-4}
244	17.039719463	17.039559451	10^{-4}	296	18.704963368	18.703338597	10^{-3}
245	17.079365817	17.078003394	10^{-3}	301	18.843979273	18.843675327	10^{-4}
246	17.113998222	17.113112319	10^{-3}	302	18.895682961	18.89237664	10^{-3}
247	17.141082423	17.132526683	10^{-2}	303	18.936286113	18.935589502	10^{-3}
248	17.184447103	17.182444113	10^{-3}	304	18.973327182	18.972060389	10^{-3}
251	17.305245615	17.297607156	10^{-2}	305	19.010908936	19.008914111	10^{-3}
252	17.331245569	17.326883903	10^{-2}	308	19.121680199	19.120984457	10^{-3}
254	17.400734608	17.400641308	10^{-4}	318	19.407004242	19.396538989	10^{-2}
255	17.454200693	17.454058463	10^{-4}	319	19.432992566	19.432246442	10^{-3}
256	17.494310641	17.49310149	10^{-3}	320	19.456230764	19.451309906	10^{-2}
258	17.547880529	17.547085193	10^{-3}				

Table 3: Better results in the case of $n = 227, 228, \dots, 320$.

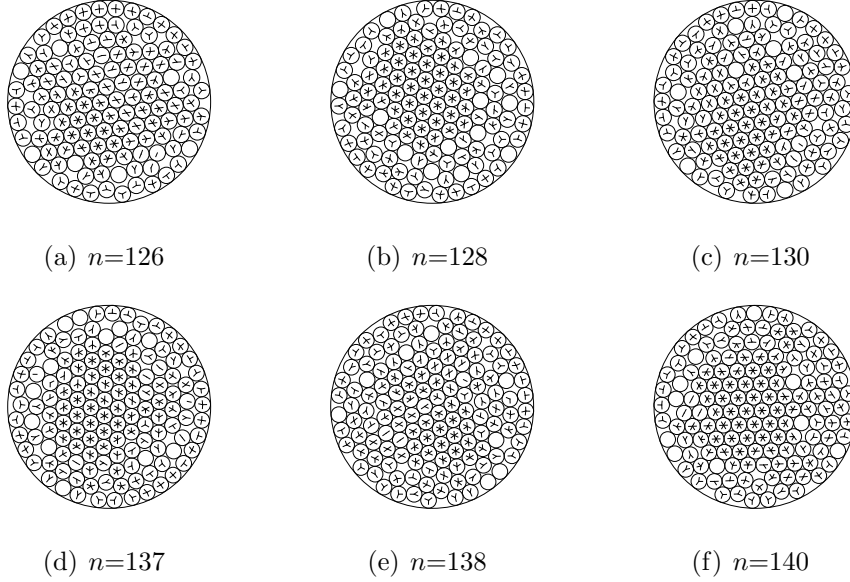


Figure 9: New layouts on some instances.

5. Conclusion

We propose an efficient quasi-physical algorithm (EQPA) for solving the classical equal circle packing problem. Our contributes include: (1) We modified the gradient descent method BFGS by only considering the neighbor structure of each circle during the gradient descent iterations. In this way, we considerably speed up the calculation, especially for larger scale of n . (2) We propose a new basin-hopping strategy of shrinking the container size and the container adjustment procedure, which are simply but very effective. Experiments on 320 ECPP instances demonstrates the success of the proposed algorithm. In the future, we will adapt the ideas for solving ECPP for unequal circles packing problem.

Reference

References

- Addis, B., Locatelli, M., Schoen, F., 2008a. Disk packing in a square: a new global optimization approach. *INFORMS Journal on Computing* 20 (4), 516–524.
- Addis, B., Locatelli, M., Schoen, F., 2008b. Efficiently packing unequal disks in a circle. *Operations Research Letters* 36 (1), 37–42.
- Akeb, H., Hifi, M., 2010. A hybrid beam search looking-ahead algorithm for the circular packing problem. *Journal of combinatorial optimization* 20 (2), 101–130.

- Akeb, H., Hifi, M., MHallah, R., 2009. A beam search algorithm for the circular packing problem. *Computers & Operations Research* 36 (5), 1513–1528.
- Akiyama, J., Mochizuki, R., Mutoh, N., Nakamura, G., 2003. Maximin distance for n points in a unit square or a unit circle. In: *Discrete and Computational Geometry*. Springer, pp. 9–13.
- Birgin, E. G., Martinez, J., Ronconi, D. P., 2005. Optimizing the packing of cylinders into a rectangular container: a nonlinear approach. *European Journal of Operational Research* 160 (1), 19–33.
- Castillo, I., Kampas, F. J., Pintér, J. D., 2008. Solving circle packing problems by global optimization: numerical results and industrial applications. *European Journal of Operational Research* 191 (3), 786–802.
- Demaine, E. D., Fekete, S. P., Lang, R. J., 2010. Circle packing for origami design is hard. *arXiv preprint arXiv:1008.1224*.
- Flores, J. J., Martínez, J., Calderón, F., 2015. Evolutionary computation solutions to the circle packing problem. *Soft Computing*, 1–15.
- Fodor, F., 1999. The densest packing of 19 congruent circles in a circle. *Geometriae Dedicata* 74 (2), 139–145.
- Fodor, F., 2000. The densest packing of 12 congruent circles in a circle. *Beiträge Algebra Geom* 41, 401–409.
- Fodor, F., 2003. The densest packing of 13 congruent circles in a circle. *Contributions to Algebra and Geometry* 44 (2), 431–440.

- Fu, Z., Huang, W., Lü, Z., 2013. Iterated tabu search for the circular open dimension problem. *European Journal of Operational Research* 225 (2), 236–243.
- Goldberg, M., 1971. Packing of 14, 16, 17 and 20 circles in a circle. *Mathematics Magazine*, 134–139.
- Graham, R., 1968. Sets of points with given minimum separation (solution to problem e1921). *Amer. Math. Monthly* 75, 192–193.
- Graham, R. L., Lubachevsky, B. D., Nurmela, K. J., Östergård, P. R., 1998. Dense packings of congruent circles in a circle. *Discrete Mathematics* 181 (1), 139–154.
- Grosso, A., Jamali, A., Locatelli, M., Schoen, F., 2010. Solving the problem of packing equal and unequal circles in a circular container. *Journal of Global Optimization* 47 (1), 63–81.
- He, K., Huang, M., Yang, C., 2015. An action-space-based global optimization algorithm for packing circles into a square container. *Computers & Operations Research* 58 (C), 67–74.
- He, K., Mo, D., Ye, T., Huang, W., 2013. A coarse-to-fine quasi-physical optimization method for solving the circle packing problem with equilibrium constraints. *Computers & Industrial Engineering* 66 (66), 1049–1060.
- Hifi, M., M'Hallah, R., 2004. Approximate algorithms for constrained circular cutting problems. *Computers & Operations Research* 31 (5), 675–694.

- Hifi, M., MHallah, R., 2006. Strip generation algorithms for constrained two-dimensional two-staged cutting problems. *European Journal of Operational Research* 172 (2), 515–527.
- Hifi, M., MHallah, R., 2007. A dynamic adaptive local search algorithm for the circular packing problem. *European Journal of Operational Research* 183 (3), 1280–1294.
- Huang, W., Li, Y., Akeb, H., Li, C., 2005. Greedy algorithms for packing unequal circles into a rectangular container. *Journal of the Operational Research Society* 56 (5), 539–548.
- Huang, W., Ye, T., 2011. Global optimization method for finding dense packings of equal circles in a circle. *European Journal of Operational Research* 210 (3), 474–481.
- Huang, W. Q., Li, Y., Jurkowiak, B., Li, C. M., Xu, R. C., 2003. A two-level search strategy for packing unequal circles into a circle container. In: *Principles and Practice of Constraint Programming–CP 2003*. Springer, pp. 868–872.
- Huang, W. Q., Li, Y., Li, C. M., Xu, R. C., 2006. New heuristics for packing unequal circles into a circular container. *Computers & Operations Research* 33 (8), 2125–2142.
- Huang, W. Q., Zhan, S. H., 1982. A quasi- physical method of solving packing problems. pp. 176–180.
- Kravitz, S., 1967. Packing cylinders into cylindrical containers. *Mathematics magazine*, 65–71.

- Lopez, C., Beasley, J., 2013. Packing unequal circles using formulation space search. *Computers & Operations Research* 40 (5), 1276–1288.
- Lü, Z., Huang, W., 2008. Perm for solving circle packing problem. *Computers & Operations Research* 35 (5), 1742–1755.
- Melissen, H., 1994. Densest packings of eleven congruent circles in a circle. *Geometriae Dedicata* 50 (1), 15–25.
- Mladenović, N., Plastria, F., Urošević, D., 2005. Reformulation descent applied to circle packing problems. *Computers & Operations Research* 32 (9), 2419–2434.
- Müller, A., Schneider, J. J., Schömer, E., 2009. Packing a multidisperse system of hard disks in a circular environment. *Physical Review E* 79 (2), 021102.
- Pirl, U., 1969. Der mindestabstand von n in der einheitskreisscheibe gelegenen punkten. *Mathematische Nachrichten* 40 (1-3), 111–124.
- Reis, G. E., 1975. Dense packing of equal circles within a circle. *Mathematics Magazine*, 33–37.
- Szabó, P. G., Markót, M. C., Csendes, T., 2005. Global optimization in geometrycircle packing into the square. In: *Essays and Surveys in Global Optimization*. Springer, pp. 233–265.
- Wang, H., Huang, W., Zhang, Q., Xu, D., 2002. An improved algorithm for the packing of unequal circles within a larger containing circle. *European Journal of Operational Research* 141 (2), 440–453.

- Ye, T., Huang, W., Lu, Z., 2013. Iterated tabu search algorithm for packing unequal circles in a circle. arXiv preprint arXiv:1306.0694.
- Zeng, Z., Yu, X., He, K., Huang, W., Fu, Z., 2015. Iterated tabu search and variable neighborhood descent for packing unequal circles into a circular container. *European Journal of Operational Research*.
- Zhang, D.-f., Deng, A.-s., 2005. An effective hybrid algorithm for the problem of packing circles into a larger containing circle. *Computers & Operations Research* 32 (8), 1941–1951.